VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Jozef Krkoška

Bakalářská práce

Vedoucí práce: Ing. Pavel Dohnálek, Ph.D.

Ostrava, 2021

Abstrakt

Táto bakalárska práca popisuje moju odbornú prax v spoločnosti profiq s.r.o. Mojou úlohou bolo zjednodušenie systému rezervácie firemných vozidiel prostredníctvom Google Sheets a jeho migrácia na užívateľsky prívetivejší informačný systém, vrátane automatizovaných testov a automatizovaného nasadenia. Práca obsahuje informácie o potrebnom teoretickom základe, jednotlivé riešenia aplikácie a potrebných funkcionalít, ďalej automatizované testy a tak isto aj automatizované nasadenie na produkčný server. V záverečnom súhrne popisujem znalosti a schopnosti, ktoré mi v priebehu odbornej praxe chýbali, dosiahnuté výsledky a ich celkové zhodnotenie.

Kľúčové slová

Flask, Python, webová aplikácia, testovanie, cloud, bakalárska práca

Abstract

This bachelor thesis describes my practice in the profiq s.r.o. company. My task was to simplify the original vehicle reservation system which used Google Sheets and its migration to more user friendly information system, including automated tests and automated deployment. This thesis contains information about the necessary theoretical basis, solutions of the assigned tasks, as well as automated tests and also automated deployment to a production server. In the final summary, I describe the knowledge and skills that I lacked during my practice, achieved results and also their overall evaluation.

Keywords

Flask, Python, web application, testing, cloud, bachelor thesis

Poďakovanie

Rád by som na tomto mieste poďakoval vedúcemu práce Ing. Pavlovi Dohnálkovi, Ph.D., konzultantovi Ing. Janovi Hájovskému a všetkým ostatným, na ktorých som sa mohol v priebehu písania práce obrátiť.

Obsah

Zoznam použitých skratiek a symbolov

SaaS	–	Software as a Service
HTML	–	Hyper Text Markup Language
GCP	–	Google Cloud Platform
SQL	–	Structured Query Language
CI/CD	–	Continuous Integration and Continuous Delivery
MVC	–	Model-View-Controller
API	–	Application Programming Interface
URL	–	Uniform Resource Locator
SSH	–	Secure Shell Protocol
AJAX	–	Asynchronous JavaScript and XML

Zoznam obrázkov

Zoznam výpisov zdrojového kódu

Kapitola 1

Úvod

Absolvoval som bakalársku prax v spoločnosti profiq s.r.o., kde som pracoval na internej webovej aplikácii na správu vozového parku. Po naštudovaní potrebných základov na tvorbu webových aplikácií som začal so zberom požiadaviek na aplikáciu od všetkých zainteresovaných osôb a s analýzou ponuky existujúcich riešení, ktoré by som mohol využiť na následnú úpravu a doplnenie funkčnosti. Keďže sa nepodarilo nájsť vhodný a rozšíriteľný základ, padlo rozhodnutie vytvoriť úplne nový informačný systém. Po postupnom spracovaní základu aplikácie a implementovaní potrebných funkcionalít bola aplikácia integrovaná do internej firemnej infraštruktúry s využitím Google cloud platformy. Aplikácia ďalej obsahuje automatizované testy, aby mohlo byť pravidelne testované, či aplikácia pracuje správne a pri nasadení zmien je otestované, či nedošlo k zmene predchádzajúcej funkčnosti.

V bakalárskej práci je najskôr predstavená spoločnosť profiq s.r.o., systém a organizácia práce v nej, a ďalej moje pracovné zaradenie v rámci študentskej stáže. V ďalšej kapitole sú popísané moje úlohy na praxi, ktoré neboli na začiatku jednoznačne zadané a vznikli až postupnými konzultáciami so zamestnancami, pre ktorých je systém určený. Postupne sú ďalej predstavené teoretické základy, ktoré si daný projekt vyžaduje ovládať. V kapitole 4 *Zvolený postup riešenia zadaných úloh* sú uvedené výsledné riešenia daných úloh aj s ukážkami funkcionality vo výslednej aplikácii. Obsahuje aj súhrn informácií o konkrétnych riešeniach na backende, frontende a v databáze. Ďalšia časť tejto kapitoly obsahuje spracovanie automatizovaných testov, tzv. *funkčné testy* a riešenie automatického nasadenia do produkcie v rámci firemnej infraštruktúry. V záverečnom súhrne popisujem znalosti a schopnosti, ktoré mi v priebehu odbornej praxe chýbali, dosiahnuté výsledky a ich celkové zhodnotenie.

Text práce rovnako obsahuje aj ukážky konkrétnych aplikačných konfigurácií, keďže som pokladal za dôležité zvýrazniť ich.

Kapitola 2

Popis odborného zamerania firmy, u ktorej študent vykonal odbornú prax a popis pracovného zaradenia študenta

Firma profiq s.r.o. [1] sa zaoberá poskytovaním služieb v oblasti softwarového vývoja a testovania, zväčša poskytuje outsourcing pre rôzne technologické spoločnosti v Spojených štátoch amerických. Ponúka buď jednotlivých špecialistov do iných tímov, alebo sa stará o celé riadenie vývoja.

Obr. 2.1: Logo spoločnosti profiq

Spoločnosť sídli v Prahe a má množstvo pobočiek, vrátane zahraničných: v Ostrave, Novom Jičíne, Martine, alebo Madride. Ostravská pobočka sa nachádza v budove podnikateľského inkubátora VŠB a trvalo zamestnáva do 50 zamestnancov, zväčša programátorov. Tí sú rozdelení do rôznych programátorských tímov, podľa projektu, na ktorom sa podieľajú. Tieto programátorské tímy sú prakticky plnohodnotne začlenené do procesu vývoja s materskými spoločnosťami a zdieľajú s nimi všetky procesy. Väčšina vývoja je organizovaná pomocou systému SCRUM, čo je metodika agilného spôsobu vývoja. V priebehu tvorby projektu sa predpokladá, že zákazníci môžu zmeniť názor na to čo potrebujú a chcú. Tím sa sústredí na maximálnu schopnosť rýchlo dodať časť s najväčšou prioritou. Práca na projekte zahŕňa zväčša každodenné online schôdzky celých vývojových tímov,

ktoré neustále konzultujú riešenie zadaných úloh, ich postupné prispôsobovanie a priorizáciu podľa aktuálnej situácie.

2.1 Projekty

Firma spolupracuje s množstvom menších aj väčších spoločností. Nižšie uvádzam niektoré z nich.

Forgerock – táto platforma ponúka kompletné riešenia na správu identít a prístupu pre podnikové, cloudové, sociálne a mobilné systémy;

Avast – softvérová spoločnosť pre kybernetickú bezpečnosť, ktorá navrhuje a vyvíja softvérové aj hardvérové riešenia bezpečnostnej infraštruktúry. Ponúka služby zabezpečenia stolných počítačov aj serverov a slúži jednotlivcom aj firmám na celom svete;

Divvypay – spoločnosť ponúka platobnú platformu pre firmy a podnikateľov, ktorá automatizuje proces vykazovania výdavkov, pomáha spoločnostiam eliminovať podvody a zbytočné výdavky a spravuje online predplatné.

2.2 Student Pool

Program Student Pool [2] je určený pre študentov VŠB, ktorí si chcú vyskúšať reálnu prácu programátora v spoločnosti. Uprednostňovaní sú obzvlášť tí študenti, ktorí sa o IT a programovanie zaujímajú aj vo voľnom čase a sú odhodlaní vzdelávať sa nad rámec školských osnov. Počas stáže títo študenti pracujú pod dohľadom senior software inžiniera na interných alebo priamo zákazkových aplikáciach. Vo firme profiq s.r.o. pracujú aj bývalí študenti VŠB, z ktorých mnohí sa do spoločnosti dostali práve cez tento program.

2.3 Moje pracovné zaradenie

Počas bakalárskej praxe som vo firme pracoval na pozícii software inžiniera. Konkrétne som sa zaoberal prácou na interných aplikáciach, najmä na aplikácii pre správu firemného vozového parku. Mojim cieľom bolo uľahčiť správu týchto vozidiel pre všetkých zamestnancov, od účtovníčky a pokladníčky, až po samotných zamestnancov, ktorí autá využívajú, a tým nahradiť pôvodný nevyhovujúci systém fungujúci prostredníctvom Google Sheets.

Kapitola 3

Úlohy zadané študentovi v priebehu praxe s vyjadrením časovej náročnosti a súpis technológií použitých na ich splnenie

Mojou úlohou na bakalárskej praxi nie je len naprogramovanie danej aplikácie, ale prakticky celý vývojový proces, od analýzy problematiky a potrebných funkcionalít, až po sprevádzkovanie serveru a automatizáciu nasadenia.

Zber požiadaviek, ktoré musí systém spĺňať, od zainteresovaných osôb – Na začiatok je treba identifikovať a kontaktovať všetky osoby, ktoré budú s daným informačným systémom pracovať. Od nich je potrebné zistiť aké konkrétne funkcionality budú od informačného systému potrebovať, ako má fungovať prístup k nim a aké majú mať vzájomné prepojenie. Odhadovaná časová náročnosť je 2 dni.

Analýza existujúcich open-source riešení – Pre zjednodušenie a zrýchlenie tvorby informačného systému existuje možnosť postaviť ho na už existujúcej platforme, ktorá by ponúkla potrebný základ a ďalšiu rozšíriteľnosť. Každé z riešení je potrebné dopodrobna preskúmať a zamyslieť sa nad možnosťami jeho následnej úpravy, ale aj nad dlhodobou udržateľnosťou a potenciálnou rozšíriteľnosťou aj o ďalšie nové funkcionality, ktoré si budúci vývoj spoločnosti môže vyžadovať. Je potrebné brať na zreteľ aj možnosť integrácie do firemnej infraštruktúry a celkovú interaktivitu a responzívnosť systému. Aj kvôli ďalším projektom, ktorým sa spoločnosť profiq venuje, sú odporúčané platformy, na ktorých by mal byť systém postavený platformy *Flask*, prípadne *Django*, ktoré využívajú návrhový vzor MVC. Odhadovaná časová náročnosť je 5 dní.

Prispôsobenie systému potrebám firmy s doprogramovaním chýbajúcej funkcionality – Na základe požiadaviek od zainteresovaných osôb je potrebné naprogramovať daný informačný

systém tak, aby poskytoval potrebnú funkcionalitu a aby bol plne integrovaný do firemnej infraštruktúry. Aplikácia by mala obsahovať funkcionality ako rezervácia auta a jej následné potvrdenie, vytváranie hlásení po jednotlivých výpožičkách, odosielanie fotiek auta, rátanie ceny za jednotlivé výpožičky a ich mesačné nacenenie pre zamestnancov, rátanie mesačných štatistík pre účtovníčku a podobne. Ďalej je nutné, aby aplikácia plnohodnotne využívala firemný prihlasovací systém založený na Google Sign-In, a teda nevyžadovala osobitnú registráciu užívateľov a zároveň neumožňovala vstup do systému užívateľom mimo firmy. Odhadovaná časová náročnosť je 35 dní.

Nasadenie na Google cloud infraštruktúre s CI/CD s automatizovanými testami – Pre zjednotenie s ostatnou firemnou infraštruktúrou je potrebné nasadiť informačný systém na Google Cloud Platform. Je nutné porovnať rôzne možnosti na tejto platforme, buď využívanie serverless služieb ako Cloud Run alebo Cloud Functions, alebo tvorba virtuálneho serveru, teda služba Virtual Machine. Treba sa zamyslieť nad možnosťami ich prispôsobenia a jednoduchosti spravovania aplikácie na nich, zároveň ale aj nad potrebným výkonom, ktorý bude aplikácia vyžadovať a z toho plynúcu najvýhodnejšiu cenu za konfigurácie jednotlivých služieb. Ďalej je potrebné vytvoriť automatizované funkčné testy, ktoré budú kontrolovať správne fungovanie všetkých dôležitých súčastí systému. Následne je nutné nakonfigurovať CI/CD nasadenie, ktoré po aktualizácii kódu programu spustí testy, po ich úspešnom vykonaní automaticky zabalí aplikáciu do Docker kontajnera a nasadí novú verziu programu na server. V prípade, že niektoré z testov neprebehnú úspešne, zabezpečí, aby na serveri ostala funkčná verzia aplikácie a teda nedošlo k jej výpadku. Je potrebné, aby tento proces prebiehal naozaj spoľahlivo a automaticky. Odhadovaná časová náročnosť je 15 dní.

3.1 Využité technológie

Pre splnenie zadania bolo potrebné naštudovať si nasledujúce technológie a spôsoby tvorby webovej aplikácie.

3.1.1 Model-View-Controller

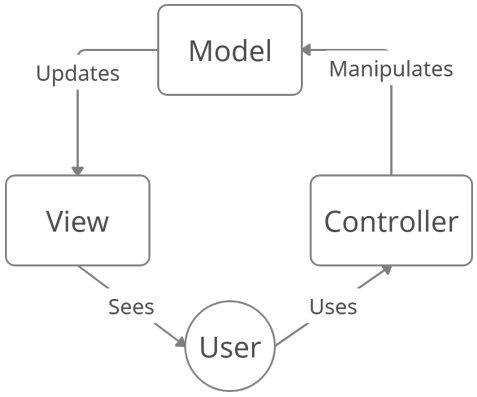
Model-View-Controller [3] je softwarový návrhový vzor najčastejšie využívaný pre tvorbu webových informačných systémov. Rozdeľuje program na tri vzájomné prepojené vrstvy - *model*, *view* a *controller*.

Model je základný komponent návrhového vzoru. Stará sa o dáta danej aplikácie a pracuje nezávisle na užívateľskom rozhraní a jeho použití.

View je vrstva, ktorú vidí užívateľ. Môže to byť tabuľka, graf alebo celý HTML kód webovej stránky. Je to reprezentácia modelu v takej forme, ktorá môže byť zobrazená užívateľovi.

Controller je vrstva, ktorá prijíma vstup od užívateľa a na základe toho odosiela požiadavky a príkazy do vrstiev *Model* a *View*.

Toto prepojenie vrstiev je znázornené na obrázku 3.1.



Obr. 3.1: MVC model

3.1.2 Flask

Jedná sa o mikro webový framework v jazyku Python [4], ktorý je postavený na architektúre MVC. V základe obsahuje len šablónovací nástroj Jinja2 [5], ktorý slúži na generovanie výsledných HTML stránok, ktoré sa odosielajú užívateľom a sadu nástrojov Werkzeug [6], ktorý sa stará o sieťovú komunikáciu. V základe neobsahuje žiadne ďalšie funkcionality, napríklad ani možnosti na prístup k databáze alebo overovanie autentifikácie. Jeho výhodou je veľká knižnica rôznych rozšírení tvorených nezávislými vývojármi, ktoré umožňujú prispôsobenie podľa vlastných potrieb. Framework *Flask* na svoj backend používajú aplikácie ako *LinkedIn*, *Reddit*, alebo *Pinterest*.

3.1.3 SQLAlchemy

SQLAlchemy je toolkit, ktorý umožňuje objektovo-relačné mapovanie a plný a flexibilný prístup k databáze [7]. Prístup k dátam umožňuje prostredníctvom jednoduchého rozhrania v jazyku Python, ktorý je nezávisle na použitom SQL engine stále rovnaký. Jedná sa o vývojársky prívetivejší prístup k databáze, než je písanie jednotlivých SQL dotazov na rôzne enginy, ktoré využívajú vlastné SQL dialekty.

3.1.4 Docker

Docker je platforma slúžiaca na virtualizáciu aplikácií, knižníc a konfiguračných súborov do takzvaných kontajnerov [8]. Tieto kontajnery sú nezávislé a štandardne izolované od ďalších kontajnerov, aj od hosťujúceho zariadenia, a teda ich môžme použiť nezávisle na použitom hardwari. Obsahujú

len potrebné dáta aplikácie bez dát operačného systému a oproti virtuálnym strojom ponúkajú menšiu veľkosť a vyšší výkon. Na jednom hardwari môže zároveň bežať viacero týchto kontajnerov a v prípade potreby môžu aj zdieľať určité zdroje alebo súbory.

3.1.5 Google Cloud Platform

Google Cloud Platform je modulárna cloud infraštruktúra od spoločnosti Google [9]. Ponúka služby ako dátové úložiská, servery, databázy a rôzne API. Sú ponúkané v rôznych konfiguráciách, v platených verziách alebo zdarma. Služba Cloud Run umožňuje hosťovanie jednotlivých Docker kontajnerov, a teda zjednodušenie pre vývojára, kedy nie je nutné spravovať celý server na ktorom daná hosťovaná aplikácia funguje. Služba Cloud SQL zas umožňuje hosťovanie databáz pre tieto aplikácie. Opačné riešenie je napríklad služba Virtual Machine, pri ktorej sa vlastne jedná o plnohodnotný virtuálny server s plným prístupom a plnou správou v rukách vývojára.

3.1.6 CI/CD

Continuous integration and continuous delivery slúži na automatizáciu testovania a nasadenia softwaru. Jedná sa o vopred daný postup inštrukcií, ktoré sa postupne vykonávajú na build serveri. Najčastejšie sú rozdelené do rôznych celkov, ktoré sa samostatne spúšťajú. Aplikácia sa v rámci nich zostaví, zabalí, vykonajú sa testy, a v prípade úspešného ukončenia všetkých častí je posledným krokom samotné nasadenie na server a spustenie novej verzie.

3.1.7 Automatizované testovanie

Automatizované testovanie [10] aplikácií je dôležité z hľadiska ich dlhodobého používania, rozširovania a udržateľnosti. Testy nám pomáhajú určiť, či jednotlivé komponenty aj aplikácia ako celok na základe vstupných parametrov generujú očakávaný výstup. Medzi základné typy testov patria:

Unit testy – Sú to menšie testy, ktoré kontrolujú, či jednotlivý komponent funguje správne. Ich výhoda je v izolácii danej časti a možnosti ju nezávisle otestovať;

Funkčné testy – Sú to väčšie testy, ktoré kontrolujú nejakú funkcionalitu ako celok, teda aj komunikáciu medzi jednotlivými komponentmi v rámci aplikácie.

S každou novou funkcionalitou aplikácie by mala byť aktualizovaná aj sada testov. Ďalej by sa mali spustiť všetky testy, aby bola istota, že omylom nebola upravená stará funkcionalita a zároveň, že nová funkcionalita funguje správne podľa očakávaní.

Kapitola 4

Zvolený postup riešenia zadaných úloh

Táto kapitola popisuje samotné detailné zadanie úloh, konkrétne riešenie aplikácie, spôsob splnenia úloh, testovanie aplikácie a jej nasadzovanie. Ďalej obsahuje nemenej dôležité konfiguračné súbory, ktoré som považoval za dôležité zvýrazniť.

4.1 Zber požiadaviek, ktoré musí systém spĺňať

Postupnými konzultáciami so zamestnancami, ktorí majú systém využívať na rezervácie, ako aj s ostatným personálom ako účtovníčka a pokladníčka, boli určené následujúce potrebné funkcionality:

- Možnosť rezervovať si vozidlo na určitý časový úsek;
- Rozlišovanie rezervácie vozidla na súkromné a pracovné účely;
- Úloha vedúceho zamestnanca potvrdiť alebo zamietnuť danú rezerváciu;
- Povinnosť zamestnancov po každom vypožičaní vyplniť formulár o ceste a o stave vozidla;
- Výpočet mesačných štatistík o využívaní vozidla, tankovaní a spotrebe;
- Možnosť tankovania v €;
- Výpočet mesačnej sumy a evidencia platieb za jazdy na súkromné účely pre jednotlivých zamestnancov;
- Rozdielny prístup do informačného systému a možnosti použitia pre jednotlivé osoby;
- Prihlásenie do systému cez firemný email prostredníctvom Google Sign-In;
- Emailové notifikácie o nových žiadostiach o rezerváciu a o schválení žiadostí.

Dané požiadavky sa ďalej upravovali a konkretizovali jednotlivé spôsoby implementácie. U poslednej funkcionality bol napokon zistený nezáujem zamestnancov o používanie, preto bola pre finálnu verziu aplikácie irelevantná.

4.2 Analýza existujúcich open-source riešení a prispôsobenie systému potrebám firmy s doprogramovaním chýbajúcej funkcionality

Pre zjednodušenie a urýchlenie tvorby informačného systému sa naskytla možnosť postaviť ho na už existujúcej platforme, ktorá by ponúkla potrebný základ a ďalšiu rozšíriteľnosť. Po dôkladnom zvážení všetkých ponúknutých možností bolo nakoniec rozhodnuté vytvoriť od základov nový informačný systém postavený na frameworku Flask. Nepodarilo sa nájsť žiadny použiteľný základ pre aplikáciu, ktorý by umožnil jednoduchú úpravu a rozšírenie bez hĺbkovej prerábky. Rozširovanie a úprava už existujúcich systémov, ktoré boli k dispozícii, by v konečnom dôsledku zabrala viac času a prostriedkov než príprava nového. Rovnako je v prípade úpravy cudzieho kódu potrebné povolenie od autora, čo by len predlžovalo a komplikovalo tento proces.

4.2.1 Spracovanie backendu

Kvôli jednoduchej práci a minimalistickosti bol na základ backendu použitý mikro framework Flask. Ten umožnil rozdeliť aplikáciu do funkčných celkov prostredníctvom tzv. blueprints.

core – obsahuje základnú funkcionalitu stránky. Stará sa o renderovanie domovskej stránky, pravidiel a užívateľských nastavení;

user – obsahuje funkcionalitu, ktorá sa stará o prihlasovanie a odhlasovanie užívateľov. Keďže má aplikácia obsahovať prihlasovanie prostredníctvom Google služieb, slúži aj na presmerovanie na API od Googlu a následné spracovanie odpovede, ktoré vyústi do prihlásenia užívateľa;

booking – stará sa o funkcionalitu spojenú so samotným požičiavaním vozidiel, teda vytváranie, schvaľovanie a mazanie žiadostí o vypožičanie auta a spracovanie finálneho hlásenia po ukončení výpožičky;

accounting – obsahuje funkcionalitu spojenú s generovaním štatistík o výpožičkách a so spracovaním a evidovaním platieb.

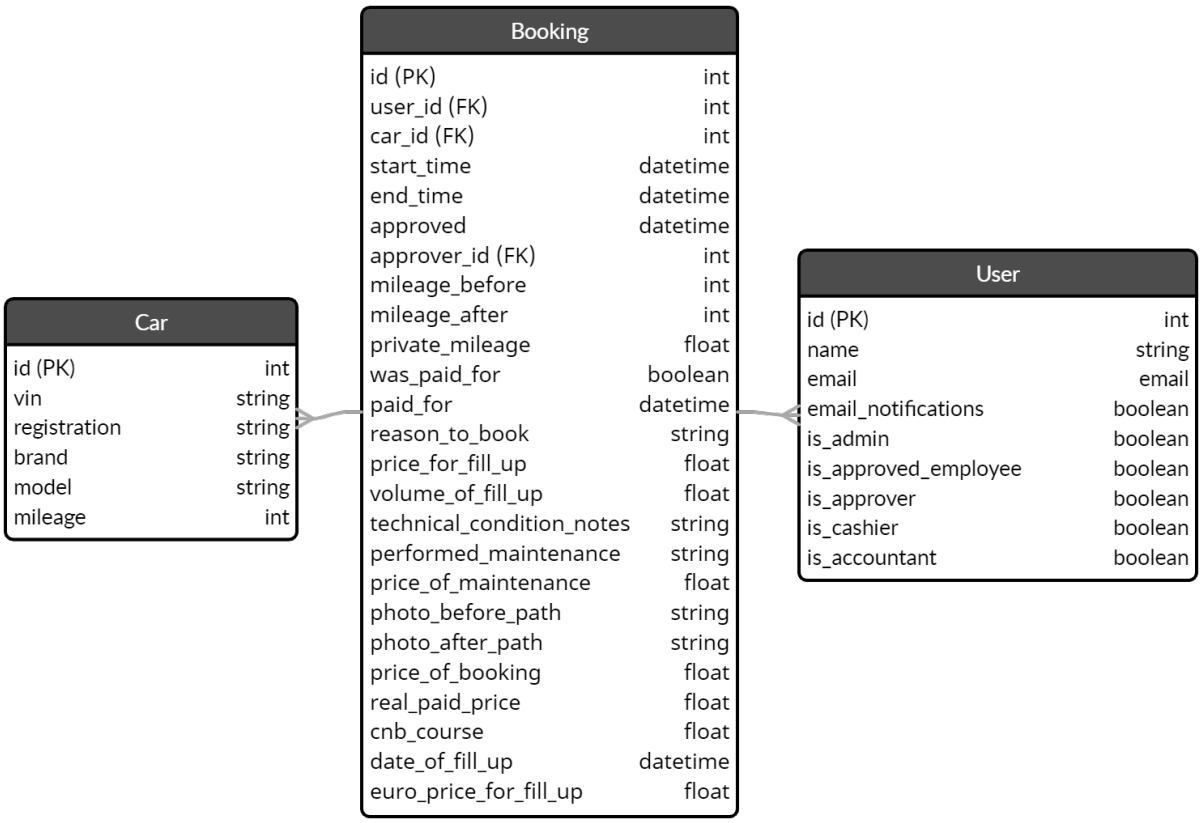
Vo frameworku Flask sa pre jednotlivé blueprinty programujú tzv. endpointy, teda funkcie, ktoré sú prístupné na určitej URL a vracajú objekt, ktorý sa zobrazí užívateľovi vo webovom prehliadači. V prípade tejto aplikácie sú to vygenerované HTML šablóny. Pri každom z endpointov sú určené ich parametre a aj to, aké typy requestov majú prijímať, a či je na prístup k nim potrebný prihlásený užívateľ.

4.2.2 Databáza

Pri tvorbe databázy sa naskytla možnosť vybrať si medzi SQL a NoSQL databázou. Po dôkladnom porovnaní výhod týchto dvoch možností, bola ako výhodnejšia zvolená SQL databáza. V prípade

tejto aplikácie totiž nebude potrebné využívať väčšinu výhod NoSQL databáz ako možnosti dynamicky meniť štruktúru databázy alebo vysokú flexibilitu schém. Práve naopak, budú využité výhody relačných databáz, kam patria rôzne spojovanie viacerých tabuliek cez ukladanie cudzích kľúčov a možnosti komplexných dopytových príkazov na úrovni databázy. Jednotlivé tabuľky a ich prepojenia sú zobrazené na obr. 4.1.

Na prístup k databáze bol použitý objektovo-relačný maper SQLAlchemy, ktorý spolu s Flaskom tvorí neoddeliteľný základ systému.



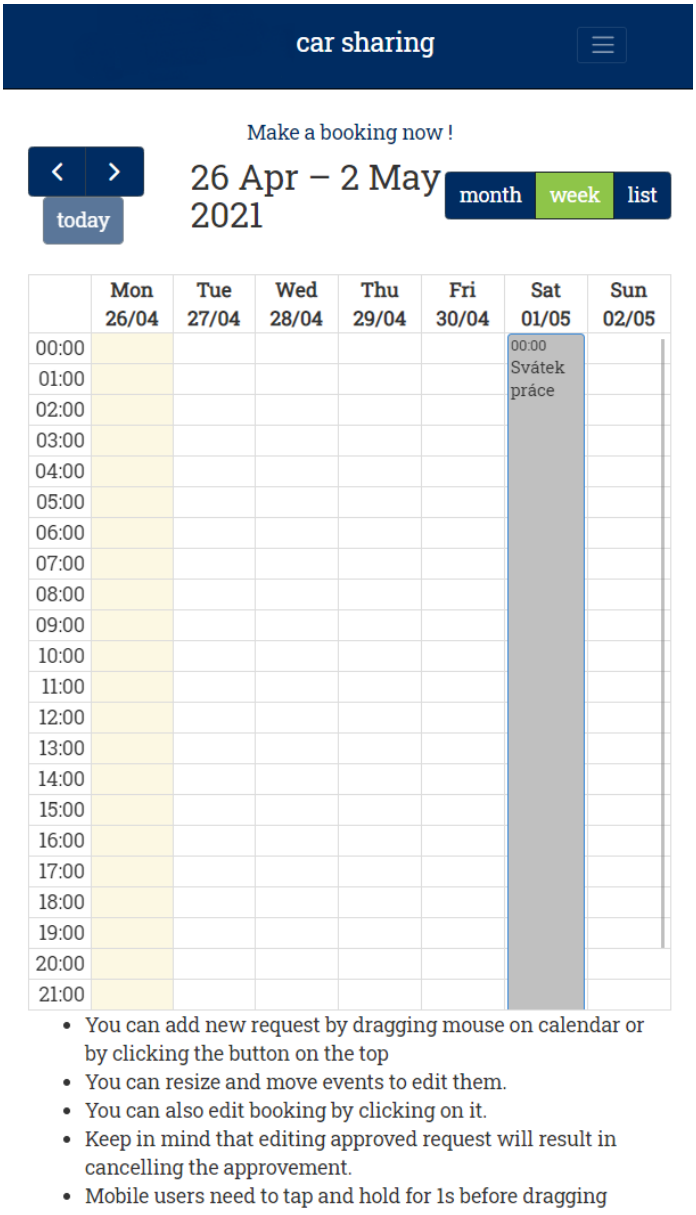
Obr. 4.1: Databázové modely

4.2.3 Frontend

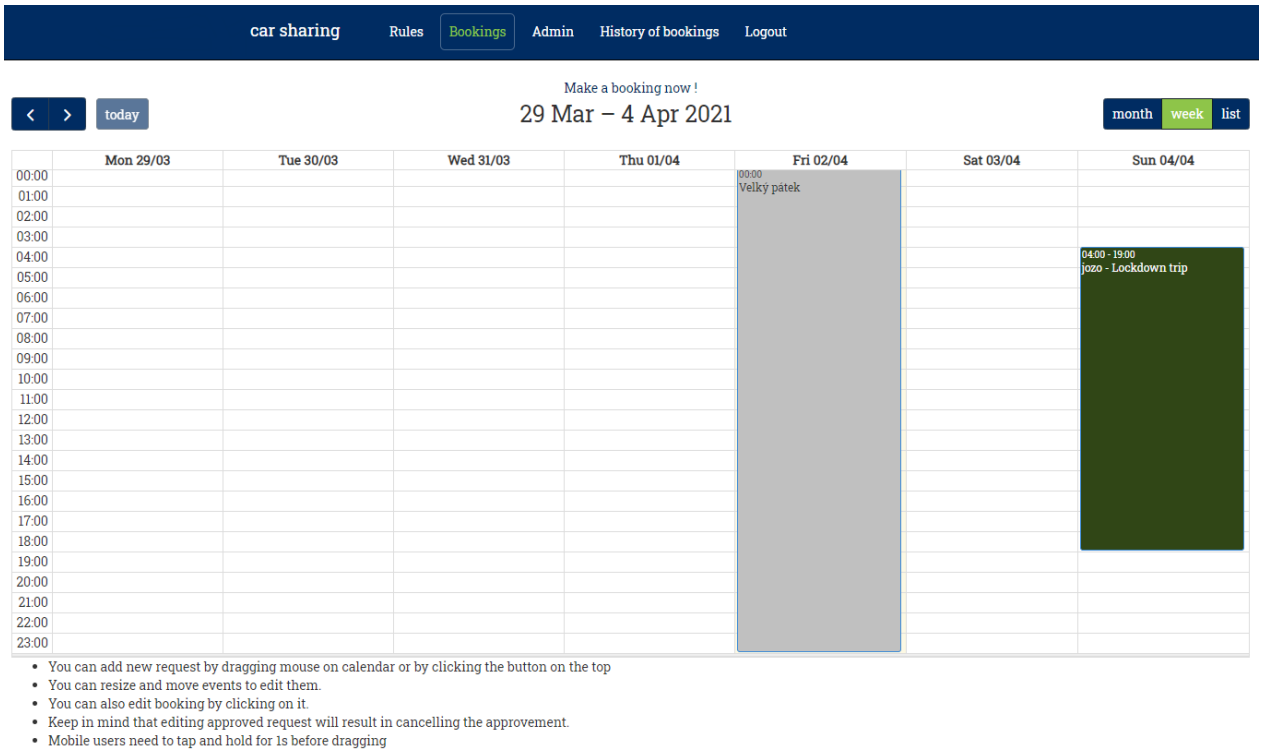
Pri tvorbe Frontendu bola použitá knižnica Bootstrap [11]. Pri návrhu tvorby aplikácie sa kvôli požiadavkám na systém rozhodlo, že je potrebné orientovať sa skôr na funkčnosť a backendovú časť než na dizajn frontendu.

K navigácii po stránke slúži navigačný bar v hornej časti obrazovky. Pre komfortnejšie využitie aplikácie bol na vytváranie žiadostí o rezerváciu použitý open-source kalendár FullCalendar [12], v ktorom užívatelia vidia aj všetky ostatné rezervácie a rovnako aj štátne sviatky. Pre vytvorenie

rezervácie stačí potiahnuť myšou po kalendári. V prípade problémov s JavaScriptom je možné vytvoriť rezerváciu aj prostredníctvom tlačidla *Make a booking now!* v hornej časti obrazovky. Kvôli každodennému využívaniu bola podmienkou aj funkčná mobilná verzia stránky, viď obrázok 4.2. Domovská obrazovka počítačovej verzie aplikácie je zobrazená na obrázku 4.3.



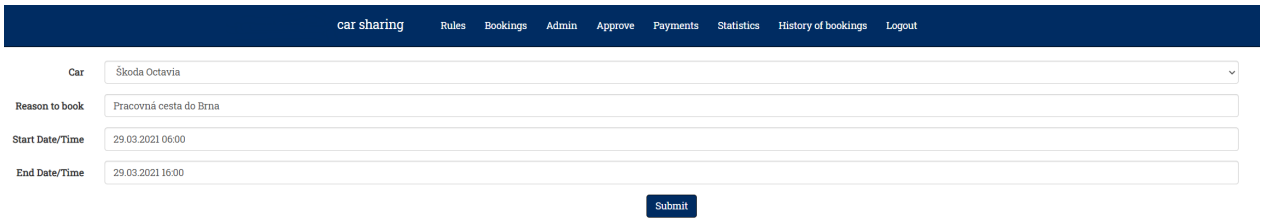
Obr. 4.2: Mobilná verzia domovskej stránky



Obr. 4.3: Hlavná stránka pre vytváranie rezervácií

4.2.4 Rozlišovanie rezervácie vozidla na súkromné a pracovné účely

Pre splnenie tohto bodu a pre celkový prehľad o dôvodoch požičiavania vozidiel bola po potiahnutí na kalendári pridaná povinnosť uvádzať dôvod rezervácie, viď obr. 4.4, v databáze uložený v poli *reason_to_book*. Toto pole sa ďalej používa aj pri rozhodovaní o priorizácii jednotlivých rezervácií, ktoré sa prekrývajú, ako uvidíme v ďalšej kapitole 4.2.5. Ďalej pri vypĺňaní hlásenia v po nej následujúcej kapitole 4.2.6 uvidíme rozdelenie prejdených kilometrov na súkromné a pracovné.



Obr. 4.4: Formulár pre pridanie rezervácie

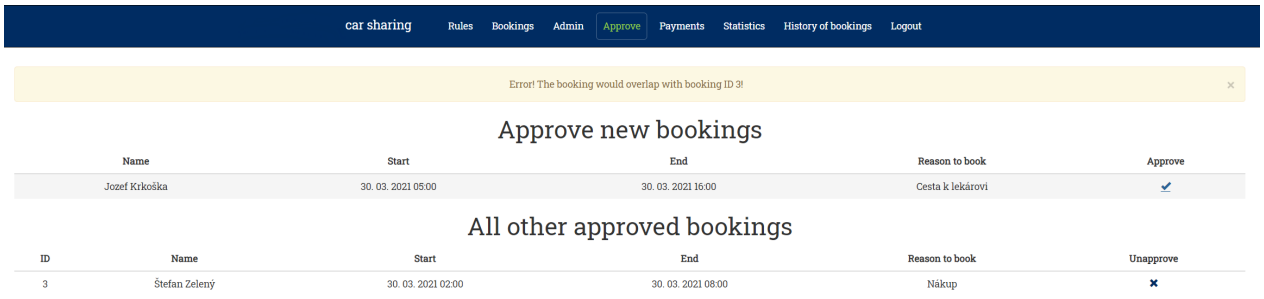
4.2.5 Úloha vedúceho zamestnanca potvrdiť alebo zamietnuť danú rezerváciu

Každá užívateľom vytvorená žiadosť o výpožičku je spočiatku evidovaná ako nepotvrdená a je potrebné ju potvrdiť vedúcim zamestnancom, v databáze má prázdne polia *approved* a *approver_id*. V prípade nepotvrdenia sa rezervácia automaticky zmaže v čase začiatku danej rezervácie. Pri potvrdení schválenia je dôležité kontrolovať, aby nebolo jedno vozidlo rezervované v rovnakom čase pre viac zamestnancov. Na získanie prekrývajúcich sa rezervácií je použitá funkcia *Bookings.get_overlaps*. Hľadajú sa rezervácie, ktoré začínajú skôr, ako končí nová rezervácia, a ktoré zároveň končia neskôr, ako začína nová rezervácia, čo môžme vidieť v následujúcej ukážke kódu:

```
def get_overlaps(self): # We use keyword "self" to access new reservation data
    return db.session.query(Booking).filter( # Querying table Booking
        and_( # Following query conditions will be joined with AND operator
            Booking.start_time < self.end_time,
            Booking.end_time > self.start_time,
            Booking.id != self.id,
            Booking.car == self.car
            Booking.approved != None))
```

Výpis 4.1: Získanie rezervácií, ktoré sa prekrývajú s novou rezerváciou

Schvaľovací formulár môžme vidieť na obrázku 4.5. Z dôvodu možného prekrývania rezervácií sú v tabuľke s už schválenými žiadosťami priamo uvádzané ID rezervácií. V prípade prekrývania rezervácií je schvaľovateľovi v žltom boxe v hornej časti obrazovky zobrazené ID rezervácie alebo rezervácií, s ktorými sa prekrýva. Následne už je na schvaľovateľovi po posúdení dôvodu výpožičky to, či novú rezerváciu potvrdí a teda zruší starú rezerváciu alebo uprednostní tú pôvodnú a novú ignoruje. Po schválení rezervácie sa do poľa *approved* vloží dátum a čas schválenia a do poľa *approver_id* ID užívateľa, ktorý danú rezerváciu schválil. V prípade zrušenia schválenia sa pole *approved* nastaví na *null* a v poli *approver_id* bude uložené ID užívateľa, ktorý danú rezerváciu zrušil.

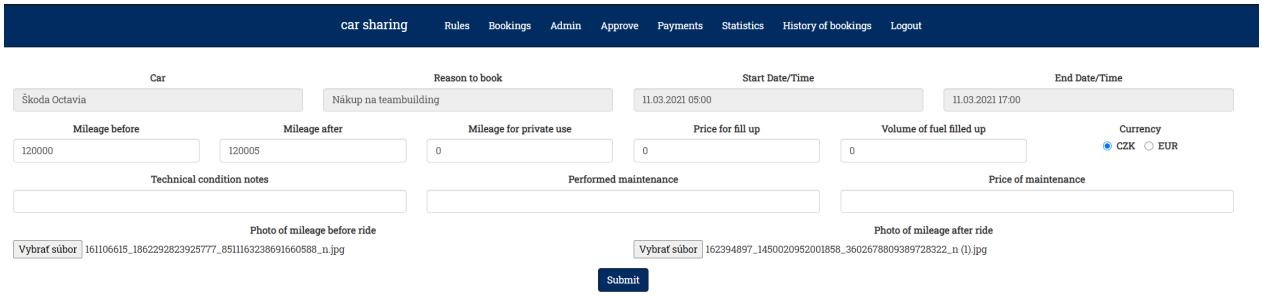


Obr. 4.5: Tabuľka pre potvrdzovanie rezervácií

4.2.6 Povinnosť zamestnancov po každom vypožičaní vyplniť formulár o ceste a stave vozidla

Po ukončení doby rezervácie sa daná rezervácia zobrazí užívateľovi na domovskej stránke ako Unfinished Booking, viď obrázok 4.6. Pre úspešné dokončenie a uzavretie rezervácie je potrebné vyplniť tzv. záverečnú správu, ako vidíme na obrázku 4.7. Je potrebné doplniť údaje pre účtovníčku a výpočet ceny pre zamestnanca. Povinným údajom je *Mileage before*, ktorý sa automaticky predvyplní podľa predošlej rezevácie, ďalej *Mileage after*, *Mileage for private use* a v prípade tankovania *Price for fill up* a *Volume of fill up*. V prípade technickej poruchy na vozidle je potrebné vyplniť údaje *Technical condition notes*, *Performed maintenance* a *Price of maintenance*. Ďalej je potrebné nahrať fotografie tachometra pred a po skončení jazdy, konkrétne do polí *Photo of mileage before ride* a *Photo of mileage after ride*. Po vyplnení potrebných polí a stlačení tlačidla *Submit* sa rezervácia pokladá za ukončenú a v prípade súkromného využitia vozidla zaň bude zamestnancovi na konci mesiaca vystavený účet, viď kapitola 4.2.8.

Obr. 4.6: Tabuľka s nedokončenými rezerváciami

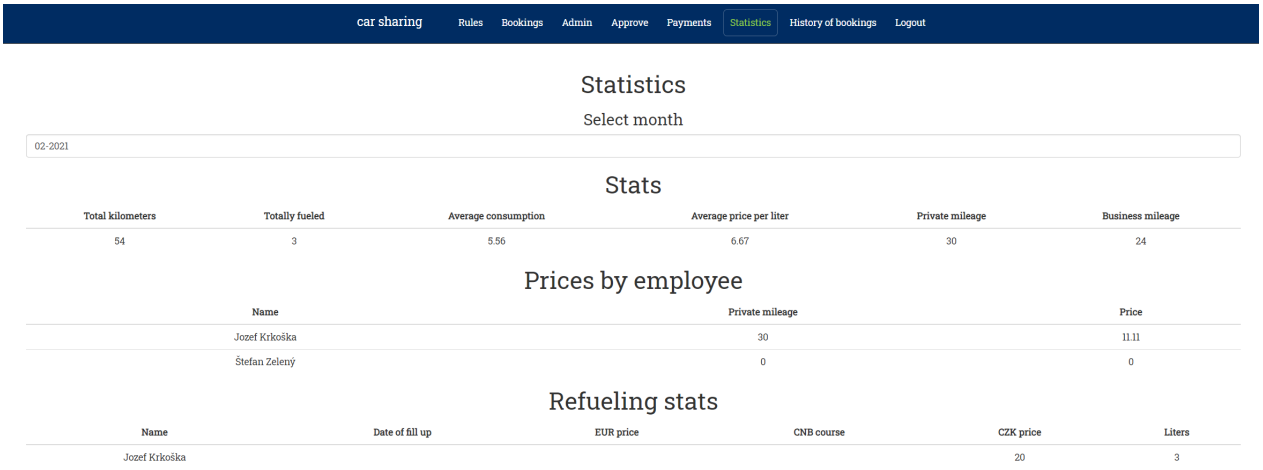


Obr. 4.7: Formulár pre dokončenie rezervácie

4.2.7 Výpočet mesačných štatistík o využívaní vozidla, tankovaní a spotrebe

Pre potreby účtovníčky sa evidujú štatistiky mesačných prejdených kilometroch, natankovaných litroch paliva, priemernej spotrebe, priemernej cene paliva za liter a celkový počet odjazdených súkromných a pracovných kilometrov na stránke *Statistics*, viď obrázok 4.8. Tieto údaje sú zobrazené v prvej z troch tabuliek, v tabuľke s názvom *Stats*. Po načítaní stránky si daný zamestnanec zvolí vo formulárovom poli mesiac a rok. Následne sa pomocou AJAX dotazu kontaktuje backend aplikácie a získajú sa z databázy potrebné dáta za daný mesiac. Tie sa pomocou jQuery pridajú do daných tabuliek, ktoré sa následne zobrazia. Vďaka tomu je daná správa tejto časti stránky dostatočne responzívna a umožňuje rýchle prepínanie medzi zvolenými mesiacmi.

Rovnako flexibilne sú načítavané aj dáta v poslednej tabuľke *Refueling stats*, ktoré sa jednoduchým dotazom získajú z databázy a následne sa zobrazia v zozname všetkých tankovaní za daný mesiac, konkrétne ich dátum, cena a objem.

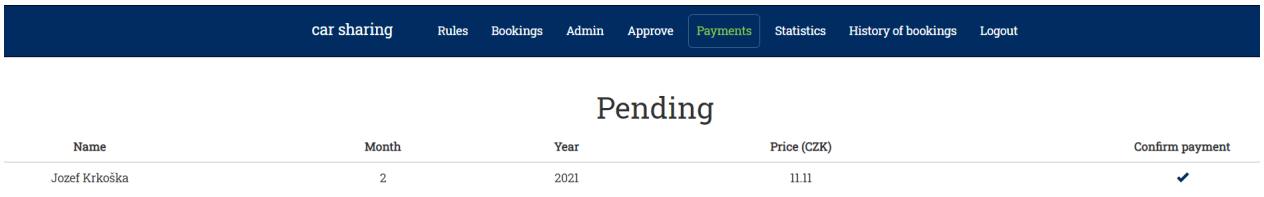


Obr. 4.8: Tabuľka štatistík a evidencie platieb

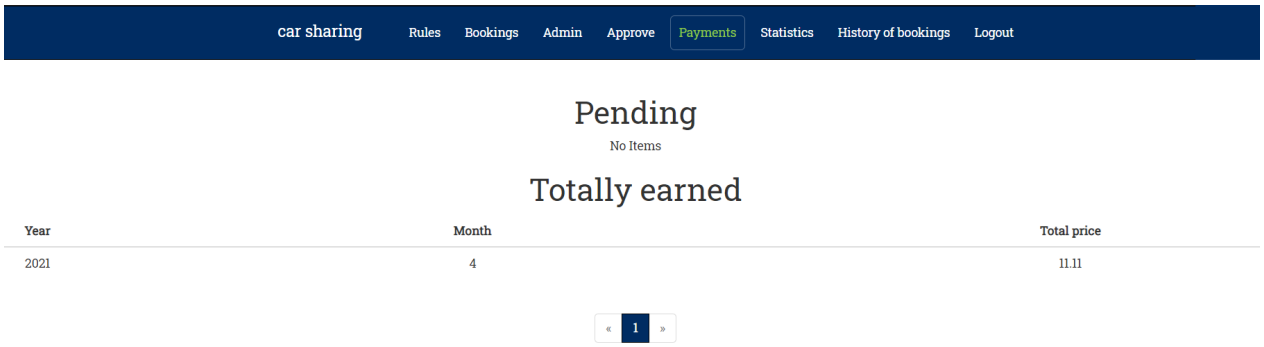
4.2.8 Výpočet mesačnej sumy a evidencia platieb za tieto jazdy pre jednotlivých zamestnancov za využívanie auta na súkromné účely

Firma sa rozhodla zamestnancom účtovať ceny za jednotlivé súkromné výpožičky podľa jasne daných pravidiel. Čím ekonomickejšie zamestnanci jazdia, tým sú ich osobné náklady na jazdy nižšie. Po skončení konkrétneho mesiaca a vyplnení záverečných správ za všetky jazdy, ktoré sa v danom mesiaci odohrali, je vypočítaná spotreba na 1 kilometer a zároveň cena za 1 liter paliva. Z týchto údajov sa jednoducho vyráta cena jazdy za 1 kilometer. Podľa tejto sumy a počtu najazdených kilometrov na súkromné účely sú následne skontrolované všetky rezervácie v danom mesiaci a tým, ktoré mali určitú vzdialenosť prejdenú na súkromné účely je priradená cena. V tabuľke *Prices by employee* na obrázku 4.8 vidíme, koľko majú jednotliví zamestnanci zaplatiť za daný mesiac.

Ďalej evidujeme aj uhrádzanie týchto platieb. Na karte Payments si môže poverený zamestnanec zobraziť súhrn čakajúcich platieb od jednotlivých kolegov. Je na vzájomnej komunikácii pracovníkov, aby si tieto pohľadávky vysporiadali. V budúcnosti by mohla pribudnúť možnosť internetových platieb, prípadne aj stiahnutia danej sumy z platu zamestnanca. Tabuľku s nezaplatenými platbami a s možnosťou potvrdiť platbu môžme vidieť na obr. 4.9. Na tejto karte ďalej poverený zamestnanec vidí aj súpis za všetky mesiace a celkovú sumu, ktorú za daný mesiac firma od pracovníkov za súkromné rezervácie inkasovala, viď obrázok 4.10.



Obr. 4.9: Tabuľka s nezaplatenými platbami



Obr. 4.10: Tabuľka s evidenciou zaplatených platieb

4.2.9 Možnosť tankovania v €

Kvôli cestám do zahraničia a možnosti tankovať v € bolo rozhodnuté o potrebe evidovania platieb za tankovanie aj v €. Tento problém bol vyriešený pridaním nepovinných údajov v záverečnom formulári, a to *Price of fill up in €*, *Date of fill up* a *ČNB course*, do ktorých užívateľ zadá cenu v €, dátum tankovania a vtedajší kurz eura podľa Českej národnej banky. Do databázy boli tieto údaje pridané ako *cnb_course*, *date_of_fill_up* a *euro_price_for_fill_up*. Podľa týchto informácií sa automaticky preráta cena do českých korún v poli *price_for_fill_up*. Toto riešenie je zvolené preto, že je kvôli účtovníctvu potrebné naďalej evidovať aj pôvodné sumy v €. V budúcnosti sa predpokladá len so zanechaním poľa s cenou v € a s dátumom tankovania, kurz eura bude získaný automaticky z internetu.

4.2.10 Rozdielny prístup do informačného systému a možnosti použitia pre jednotlivé osoby

Pre účel rôznych užívateľských práv sú v databáze užívateľov následujúce parametre:

is_admin – Umožňuje prístup do admin interface, kde daný administrátor pracuje priamo so všetkými údajmi v databáze;

is_approved_employee – Slúži na evidovanie, či daný zamestnanec prešiel školením vodičov a môže si vytvárať žiadosti o rezerváciu, užívateľom bez tohto povolenia nie je vôbec umožnený prístup do aplikácie;

is_approver – Dáva užívateľovi právo schvaľovať a rušiť schválenie žiadostí o rezerváciu;

is_cashier – Umožňuje užívateľovi prístup na stránku Payments, kde potvrdzuje prijatie jednotlivých platieb od zamestnancov a ďalej na stránku Statistics;

is_accountant – Dáva prístup na stránku Statistics a History of Bookings, kde sú všetky údaje z databázy o každej z rezervácií.

Na základe týchto práv sú prihláseným užívateľom zobrazované možnosti v navigačnej lište. Práva sú kontrolované aj jednotlivo pri každom vstupe na stránku, ktorá práva vyžaduje a pri každej akcii, ktorá vyžaduje zásah do databázy.

4.2.11 Prihlásenie do systému cez firemný email prostredníctvom Google Sign-In

Pre integráciu s ostatnou firemnou infraštruktúrou bolo potrebné aj do tejto aplikácie pridať prihlasovanie prostredníctvom Google Sign-In. K tomu bola využitá oficiálna API od spoločnosti Google pre jazyk Python, ktorá je volaná prostredníctvom knižnice *oauthlib*. Bolo potrebné na Google Cloud Platform založiť *OAuth* token, ktorý je následne dodaný aplikácii v rámci nasadenia pomocou premenných prostredia, viď kapitola 4.3.5. Užívateľ je po stlačení tlačidla *Login* presmerovaný na Google API, ktorá aplikácii v prípade úspešného prihlásenia vráti informácie o prihlásenom užívateľovi.

4.3 Nasadenie na Google cloud infraštruktúre s CI/CD s automatizovanými testami

Pre zjednotenie s ostatnou firemnou infraštruktúrou bolo potrebné nasadiť informačný systém na Google Cloud Platform. Ďalej bolo nutné vytvoriť automatizované testy a CI/CD nasadenie, ktoré po aktualizácii programu spustí testy a po ich úspešnom vykonaní automaticky nasadí novú verziu aplikácie na server.

4.3.1 Testy

Na správne otestovanie funkčnosti a všetkých dôležitých prípadov použitia pri každom nasadení bolo žiadané pridať do aplikácie automatizované testy. Na spúšťanie testov bola vybraná knižnica nosetests [13], ktorá využíva zabudovaný framework na testovanie unittest [14]. Každý z testov je spustiteľný a vykonateľný samostatne, bez závislosti na ostatných testoch. Rovnako treba zdôrazniť, že po každom teste musí byť databáza v pôvodnom stave, aby sa neovplyvňovali nasledovné testy. Podľa toho, aké funkcionality aplikácie sa testujú, boli tieto testy rozdelené do jednotlivých tried.

Pre správne fungovanie testov a testovacieho frameworku museli byť testy uložené vo vopred danej súborovej štruktúre. Všetky museli byť umiestnené v zložke /tests, ktorá bola ďalej rozdelená

na viac priečinkov podľa funkčných celkov. Do nich boli umiestnené súbory s jednotlivými testovacími triedami. Názov každého zo súborov musel začínať slovom "test", rovnako ako názov každej testovacej funkcie v daných triedach. Triedy museli dediť z triedy TestCase z knižnice flask_testing [15].

Pre každú z testovacích tried bolo potrebné správne implementovať nasledovné metódy:

create_app – Táto metóda sa spúšťa práve raz, vždy pri začatí testov v konkrétnej triede. Slúži teda na počiatočnú konfiguráciu prerekvizít potrebných na jednotlivé testy;

set_up – Metóda setUp sa spúšťa pred každým z vykonávaných testov a slúži na prípravu dát, s ktorými sa v testoch pracuje;

tear_down – Táto metóda sa naopak spúšťa po ukončení každého testu pre vyčistenie dát z databázy a umožnenie spustenia ďalších testov.

Ďalej uvádzam príklad jednoduchej testovacej triedy Usertest, ktorá testuje prístup do aplikácie autorizovaným a neautorizovaným užívateľom.

```
class UserTest(TestCase):
    # Turning off rendering HTML templates which are not needed for tests
    render_templates = False

    def create_app(self):
        # Setting the application to test mode
        app.config['TESTING'] = True
        # Setting up test database
        app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:////var/car-app/db/tests.
            db'
        # Deleting original database from cache memory
        db.session.remove()
        return app

    def set_up(self):
        # Creating new database and saving into cache memory
        db.create_all()

    def tear_down(self):
        # Deleting database from cache memory and from the disk
        db.session.remove()
        db.drop_all()
```

```
    # Function that checks whether the user has access to application without
        approval
    def test_first_login(self):
        # We send data about a user who is not in the database to a function that
            process a valid response from the Google API.
        process_google_response({'email':'Testing2@profiq.com', 'name':'Testing2'
            })
        # We try to access the page for creating new bookings
        self.client.get(url_for('booking_blueprint.bookings'))
        # Check whether we were redirected to unauthorized error page
        self.assert_template_used('user/unauthorized.html')

    # A function that tests authorized user login to a session
    def test_approved_user_login(self):
        # Create a new user in database with access to application
        user = User(name='Testing', email='Testing@profiq.com',
            is_approved_employee=True)
        db.session.add(user)
        db.session.commit()
        # We send data about a user who is in the database and is approved to a
            function that process a valid response from the Google API.
        process_google_response({'email': 'testing@profiq.com'})
        # Check whether the user was logged in into session
        assert current_user.email == 'testing@profiq.com'
```

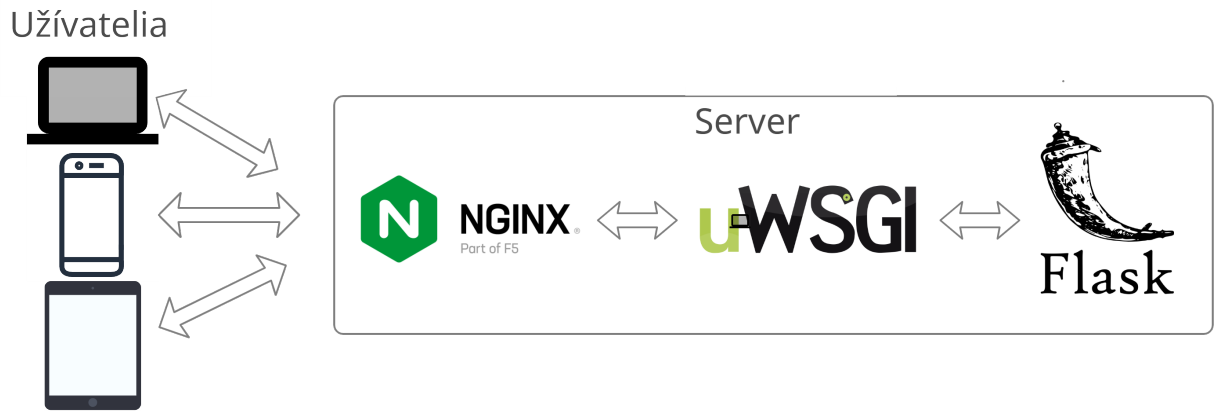
Výpis 4.2: Príklad testovacej triedy UserTest

4.3.2 Produkčný server

Keďže framework Flask v základe obsahuje len vývojársky server, na ostré hosťovanie aplikácie bol použitý autormi Flasku odporúčaný server uWSGI [16]. Ten sa stará o beh aplikácie v bezpečnom prostredí a zároveň zvláda prijímať viac užívateľských požiadaviek súčasne.

Keďže je nutné, aby aplikácia bežala v bezpečnom prostredí a zároveň bola aj prístupná z internetu, žiadalo sa na prístup k stránke využitie šifrovaného protokolu HTTPS, teda pomocou portu 443. K tomuto účelu bol medzi serverom uWSGI a prístupom z internetu využitý server NGINX [17]. Tieto servery spolu komunikujú a vymieňajú si dáta pomocou socketov. Toto prepojenie ilustruje obrázok 4.11.

Pre kompatibilitu, súčasný beh a správne poradie štartovania týchto serverov bola využitá aplikácia supervisord [18], ktorá slúži na monitorovanie a kontrolu aplikácií spustených na pozadí.



Obr. 4.11: Konfigurácia serveru

4.3.3 Docker

Aby bola aplikácia jasne izolovaná od zariadenia na ktorom je hosťovaná, používame aplikáciu Docker. Podľa jasne definovaného postupu vytvorí po každom novom ubalení aplikácie nové izolované prostredie, tzv. Docker kontajner, pričom využíva jeden zo zabudovaných obrazov systému, v našom prípade s nainštalovaným Pythonom 3.8. Zabráni sa tak problémom spôsobených závislosťami na hosťujúcom zariadení. Postup na ubalenie kontajnera je uložený v súbore Dockerfile.

```
# Setting system image, we use the one with Python 3.8
FROM python:3.8

# Installing system dependencies
RUN apt-get update
RUN apt-get install -y --no-install-recommends libatlas-base-dev gfortran nginx
    supervisor
RUN pip3 install uwsgi

# Creating new user "nginx" which will run the server
RUN useradd --no-create-home nginx

# Deleting default configurations
RUN rm /etc/nginx/sites-enabled/default
RUN rm -r /root/.cache

# Copy configurations into system folders
COPY server-conf/nginx.conf /etc/nginx/
```

```
COPY server-conf/flask-site-nginx.conf /etc/nginx/conf.d/
COPY server-conf/uwsgi.ini /etc/uwsgi/
COPY server-conf/supervisord.conf /etc/

# Setting the working directory and copying program files
WORKDIR /var/car-app/app
COPY . .

# Installing Python dependencies
RUN pip3 install -r requirements.txt

# Opening port 443 for HTTPS
EXPOSE 443

# Turning on the supervisord application
CMD ["/usr/bin/supervisord"]
```

Výpis 4.3: Konfigurácia ubalenia aplikácie v súbore Dockerfile

Zároveň bolo potrebné aplikácii sprístupniť niektoré súbory mimo Docker kontajneru. Dôležitá je napríklad databáza, ktorú chceme uchovať aj po aktualizácii a novom skompletovaní Docker kontajnera. Na toto slúži súbor *docker-compose.yml*, v ktorom je nastavený aj súbor s premennými prostredia alebo preklad portov medzi hostujúcim zariadením a Docker kontajnerom.

4.3.4 Server na Google Cloud Platform

Na hosťovanie aplikácie bolo najmä kvôli cene rozhodnuté využiť službu Google Virtual Machine, teda virtuálny server, ku ktorému existuje jednoduchý prístup pomocou protokolu SSH. Tento server je v základnej nízkej konfigurácii k dispozícii na Google Cloud Platform zdarma. Zároveň bolo preto rozhodnuté nevyužiť Google SQL server, ktorý by vyžadoval dodatočné náklady. Je využitá asi najjednoduchšia SQL databáza SQLite priamo vo Virtual Machine. Táto databáza ponúka dostatočnú robustnosť pre účely tohto projektu a hosťujúci, pomerne málo výkonný server vyťažuje len minimálne. Keďže bola využitá knižnicu SQLAlchemy, zmena databázového enginu by v prípade potreby spočívala len v zmene v konfiguračnom súbore a premigrovaní dát v databáze.

Keďže boli využité Docker kontajnery, server potreboval len minimálnu konfiguráciu na spustenie aplikácie. Najprv bolo potrebné v systéme vytvoriť užívateľa gitlab.ci, ktorý bude neskôr využitý na nasadenie pomocou Gitlab CI/CD. Ďalej bola vytvorená zložka /var/car-app/ a tomuto užívateľovi boli dané všetky práva v nej. Podľa konfigurácie sú práve tu ukladané aplikačné súbory, databáza a odoslané fotografie áut zo záverečných hlásení. Ako posledné bolo potrebné vytvoriť SSH kľúč, ktorý

je neskôr použitý pri nasadení prostredníctvom Gitlab CI/CD na pripojenie na server a aktualizáciu aplikácie.

4.3.5 Automatické nasadenie s testami

Na automatizáciu spúšťania testov a nasadenia aplikácie bolo rozhodnuté využiť nástroj Gitlab CI/CD. Ten umožňuje vytvoriť rôzne etapy nasadenia, ktoré sa samostatne vykonávajú, pričom všetky sa musia ukončiť úspešne. Skripty v týchto etapách sa spúšťajú na Gitlab serveroch. V každom je osobitne vybrané, na akom obraze systému majú dané skripty bežať, aké prerekvizity sa majú odohrať pred spustením a ďalej príkazy na samotné spustenie skriptu. Toto nasadenie sa vykoná vždy po odoslaní novej zmeny do hlavnej vetvy na Gitlabe. Nasadenie tejto aplikácie bolo rozdelené na 2 časti:

- Tests;
- Deploy.

```
Tests:
  stage: unittest
  image: debian:latest
  before_script:
    # A set of commands that prepares the prerequisites for running tests. We need
        to create a folder for the application and database, and install the nose
       application
    - mkdir /var/car-app /var/car-app/db
    - apt-get update
    - apt-get install python3-pip -y
    - pip3 install -r requirements.txt
    - pip3 install nose
    - cd tests
  script:
    # Run the tests
    - nosetests
```

Výpis 4.4: Testovacia etapa nasadenia

V prípade úspešného ukončenia testov sa automaticky prejde na ďalšiu etapu, t. j. samotné nasadenie na server. Pri nasadení je potrebné sa pomocou protokolu SSH pripojiť na server a vykonať tam zmeny. Pri pripájaní na server sa vychádzalo z dokumentácie Gitlab Docs [19]. Ďalej bolo potrebné nastaviť na Gitlabe skryté premenné pre nasadenie, ktoré niesu priamo viditeľné v

kóde. Deje sa to kvôli zabezpečeniu a možnej zmenu týchto parametrov bez úpravy kódu, jedná sa napríklad o IP adresy alebo heslá.

```
Deploy:
  stage: deploy
  image: debian:latest
  before_script:
    # Preparing to connect via SSH according to Gitlab documentation. Firstly we
       will install ssh-agent which takes care of work with SSH keys.
    - 'command -v ssh-agent >/dev/null || ( apt-get update -y && apt-get install
       openssh-client -y )'
    - eval $(ssh-agent -s)
    # Add the SSH private key from the environmental variables to the SSH key
       database and set the correct access rights
    - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    # Add the IP address of server from the environmental variables to the known
       addresses and set the correct access rights
    - ssh-keyscan $SERVER_IP >> ~/.ssh/known_hosts
    - chmod 644 ~/.ssh/known_hosts
  script:
    # Add the necessary environmental variables to run the application from deploy
        variables to the sshenv file
    - echo "GOOGLE_CLIENT_ID=$GOOGLE_CLIENT_ID" > sshenv
    - echo "GOOGLE_CLIENT_SECRET=$GOOGLE_CLIENT_SECRET" >> sshenv
    - echo "DEFAULT_ADMINS=$DEFAULT_ADMINS" >> sshenv
    # Copy all application data to server via SCP protocol
    - scp -rp "${PWD}" gitlab.ci@$SERVER_IP:~
    # Copy the file with sshenv variables to server
    - scp sshenv gitlab.ci@$SERVER_IP:~/car-app/.env
    # Delete old docker images and start building new docker container on server
    - ssh gitlab.ci@$SERVER_IP "docker image prune --force && cd car-app && docker
       -compose up --detach --build"
```

Výpis 4.5: Etapa nasadenia na Google server

Týmto bol proces nasadenia kompletne zautomatizovaný. Po aktualizácii programu na Gitlabe sa automaticky vykonajú všetky testy a nová verzia sa nasadí na vzdialený server. Celý tento proces trvá niekoľko minút a v konečnom dôsledku je aplikácia neprístupná len niekoľko sekúnd.

Kapitola 5

Záver

Počas bakalárskej praxe sa podarilo naprogramovať funkčnú verziu aplikácie podľa zadaných požiadaviek, ktorá vyhovuje zamestnancom, ktorí ju využívajú ako na požičiavanie auta, tak aj na administráciu tohto procesu. Rovnako boli úspešne vytvorené automatizované testy najdôležitejších vlastností a funkcionalít systému, ktoré pomôžu zabezpečiť bezchybnú funkcionalitu aj v prípade úprav kódu. Aplikácia na svoj chod využíva jednotnú firemnú infraštruktúru na Google Cloud Platform, na ktorú je nahrávaná pomocou CI/CD. Zjednodušuje a automatizuje tak celý proces od programovania až po samotné reálne využitie nových funkcionalít.

5.1 Teoretické a praktické znalosti a zručnosti získané v priebehu štúdia uplatnené študentom v priebehu odbornej praxe

Pri tvorbe webovej aplikácie v jazyku Python som čerpal znalosti najmä z predmetu *Skriptovací programovací jazyky a jejich aplikace*, v ktorom bola okrem programovacieho jazyka Python zahrnutá aj tvorba backendu pre webovú aplikáciu.

Na tvorbu frontendu webovej aplikácie v jazyku JavaScript a HTML som zase využil znalosti z predmetu *Tvorba aplikací pro mobilní zařízení 1*, ktorý zahŕňal všetky potrebné aspekty tvorby frontendu pre rôzne použitie. Naučil ma pracovať aj s knižnicami jQuery a Bootstrap.

Na postupnú agilnú tvorbu aplikácie, jej zálohovanie a verzovanie som používal verzovací systém git, ktorého základy sme využívali pri tvorbe projektu v predmete *Tvorba aplikací pro mobilní zařízení 2*.

V práci som na základe odporúčaní pracoval v operačnom systéme Linux, ktorého základy mi ponúkol predmet *Správa operačních systémů*. Zároveň mi tento predmet pomohol aj pri záverečnej automatizácii nasadenia aplikácie, ktorého proces sa odohráva na Linuxovej distribúcii.

Behom tvorby aplikácie som mnohokrát využíval rôzne tutoriály a internetové fóra, ktoré boli prevažne v angličtine. Technickú slovnú zásobu v tomto jazyku som nadobudol v predmetoch *Ab/I-FEI - Ab/IV-FEI*.

5.2 Znalosti či zručnosti chýbajúce študentovi v priebehu odbornej praxe

Absentovali mi najmä skúsenosti s testovaním aplikácií, pri ktorých som spočiatku neovládal, ako detailne má byť daná funkcia otestovaná, a aké elementárne majú dané testy byť.

Rovnako mi značnú chvíľu zabralo aj zoznámenie sa s cloudovou aplikáciou Google Cloud Platform a prácou so serverom pomocou vzdialeného serveru. Myslím, že v dnešnej dobe by bolo užitočné, viac sa zamerať aj na tieto aspekty tvorby softwarového inžinierstva.

5.3 Dosiahnuté výsledky v priebehu odbornej praxe a ich celkové zhodnotenie

Celoročným výsledkom mojej bakalárskej práce je funkčná, firmou naďalej využívaná aplikácia, ktorá zamestnancom výrazne zjednodušila proces požičiavania vozidiel. Najväčším prínosom bakalárskej práce pre mňa bola skúsenosť s novým, rozsiahlejším projektom. Vyskúšal som si, aké je pracovať na softwari na základe užívateľských požiadaviek, proces od analýzy potrebných funkcionalít, cez samotný vývoj, až po automatizované nasadenie aplikácie. Naučil som sa pracovať s technológiami pre automatické testovanie a s cloudovými službami. V neposlednom rade som si prehĺbil úroveň komunikácie s kolegami a nadobudol skúsenosti pri práci vo firemnom prostredí.

Použitá literatúra

1. *profiq* [online] [cit. 2021-02-01]. Dostupné z: `https://www.profiq.com/`.
2. *Student Pool* [online] [cit. 2021-02-01]. Dostupné z: `https://www.pracujprosiliconvalley.cz/student-pool/`.
3. *MVC* [online] [cit. 2021-02-03]. Dostupné z: `https://www.codecademy.com/articles/mvc`.
4. GRINBERG, Miguel. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, 2018.
5. *Jinja* [online] [cit. 2021-02-09]. Dostupné z: `https://jinja.palletsprojects.com/en/2.11.x/`.
6. *Werkzeug* [online] [cit. 2021-02-09]. Dostupné z: `https://palletsprojects.com/p/werkzeug/`.
7. MYERS, Jason. *Essential SQLAlchemy: Mapping Python to Databas*. O'Reilly Media, 2015.
8. TURNBULL, James. *The Docker Book: Containerization is the new virtualization*. Independently published, 2014.
9. GEEWAX, JJ. *Google Cloud Platform in Action*. Manning Publications, 2018.
10. AXELROD, Arnon. *Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects*. Apress, 2018.
11. JAKOBUS, Benjamin. *Mastering Bootstrap 4: Master the latest version of Bootstrap 4 to build highly customized responsive web apps*. Packt Publishing, 2018.
12. *FullCalendar* [online] [cit. 2021-02-20]. Dostupné z: `https://fullcalendar.io/`.
13. *Nose* [online] [cit. 2021-02-10]. Dostupné z: `https://nose.readthedocs.io/en/latest/`.
14. *UnitTest* [online] [cit. 2021-02-15]. Dostupné z: `https://docs.python.org/3/library/unittest.html`.
15. *Flask-Testing* [online] [cit. 2021-02-23]. Dostupné z: `https://pythonhosted.org/Flask-Testing/`.
16. *uWSGI* [online] [cit. 2021-02-15]. Dostupné z: `https://uwsgi-docs.readthedocs.io/en/latest/`.

17. *NGINX* [online] [cit. 2021-02-16]. Dostupné z: `https://www.nginx.com/`.

18. *Supervisord* [online] [cit. 2021-02-16]. Dostupné z: `http://supervisord.org/`.

19. *Gitlab Docs* [online] [cit. 2021-02-20]. Dostupné z: `https://docs.gitlab.com/ee/ci/ssh_keys/`.